

A Sequence-Focused Parallelisation of EMBOSS on a Cluster of Workstations

Karl Podesta, Martin Crane, and Heather J. Ruskin

School of Computing, Dublin City University, Ireland
kpodesta,mcrane,hruskin@computing.dcu.ie

Abstract. A number of individual bioinformatics applications (particularly BLAST and other sequence searching methods) have recently been implemented over clusters of workstations to take advantage of extra processing power. Performance improvements are achieved for increasingly large sets of input data (sequences and databases), using these implementations. We present an analysis of programs in the EMBOSS suite based on increasing sequence size, and implement these programs in parallel over a cluster of workstations using sequence segmentation with overlap. We observe general increases in runtime for all programs, and examine the speedup for the most intensive ones to establish an optimum segmentation size for those programs across the cluster.

1 Introduction

Probably the most popular sequence analysis tool in present use is BLAST [1], a program for aligning sequences of biological data to investigate similarity. Single or multiple sequences form a query, which is searched against a database of sequences using a heuristic algorithm. The returned output is a list of matching target sequences. This whole process (a job) is repeated many thousands of times a day on publicly available BLAST servers. A number of parallel implementations [8, 10, 12–14] of the tool have been investigated and proposed at the levels of sequence, query, database, and job, to improve performance. Additionally, a number of other sequence analysis tools and implementations [3, 6] (mostly alignment-based) have also been proposed which exploit parallelism at these levels.

In this work, we investigate if these methods can be applied to a wider range of tools (namely the EMBOSS suite [7]), and focus on the performance of the suite for increasing sizes of sequence. We present a parallel implementation that splits sequences into smaller ones (with an overlap to allow examination of the area at the split), and distributes these sub-sequences to a number of machines (or 'nodes') in a cluster of workstations.

The paper is organised into 3 major sections. The following section (Section 2) provides some background information on EMBOSS, along with a discussion of related work in parallel implementations of other programs. Section 3 presents a serial (non-parallel) analysis of programs in the EMBOSS suite. This

is in order to examine the effect of increasing the size of input sequence data, and to identify potential opportunities for parallelisation, if any exist. A classification of programs is discussed, along with methodology and results. Lastly, Section 4 presents a parallel implementation, and an examination of results for the more intensive programs in the EMBOSS suite. A final section presents our conclusions.

2 Background

2.1 EMBOSS

EMBOSS is a suite of programs (presently 160 programs as of release 2.8.0) used in molecular biology, whose primary function is to analyse sequences of biological data. The suite is freely available under the GNU Public License, is open source, and is maintained and developed at the Rosalind Franklin Centre for Genome Research (RFCGR) in Hinxton, Cambridge in the United Kingdom.

Programs are accessible through a number of methods (command line, web interface, and graphical user interface), and the suite is currently installed at, and supported by, a number of sites throughout EMBnet, the European network of molecular biologists, as well as a number of other sites around the world.

Although initially developed as a free replacement for a commercial software package, EMBOSS has a range of applications that offer a wide functional variety [4, 15] capturing many common tasks of the bioinformatics specialist [9]. For example, programs in EMBOSS can be used to locally or globally align DNA sequences based on a number of techniques, predict the 2-dimensional structure of a protein, or calculate evolutionary distances between sequences in a multiple alignment. In addition, a wide number of data formats are supported and many public sequence databases (locally installed) can be used as sources of sequence data. Finally, there are also programs for displaying and editing sequences within the suite. A more complete classification of programs will be discussed in Section 3.2.

Further to the programs included in the suite is an Application Programmers Interface (API). This API provides core functions that can be used by programmers to create their own EMBOSS applications, and those applications of general benefit can be submitted for possible inclusion in the suite. Examples of core functions provided include those for reading and writing sequences in different formats, display functions, math functions, and string manipulation functions. All programs have a consistent interface, which allows programs to be embedded in batch runs, or attached to different interfaces with ease, thanks to the API. Importantly for this study, this facilitates a suite-wide analysis using common input sequences and parameters.

2.2 Related Work

Previous analyses of sequence analysis methods and programs have exploited parallelism at different levels, focused on decomposition of major components of

the operation and on the operation itself. In [8], three levels of parallelisation within BLAST are proposed; fine grained (splitting a query), medium grained (splitting a database), and coarse grained (splitting jobs). Only the last is further investigated, and uses the Portable Batch System (PBS) [2] to distribute BLAST jobs across a cluster of 25 CPUs. Other implementations that split BLAST jobs are detailed in [10] and [14], which use GNU Queue [16] and a custom Perl script to manage jobs, respectively.

At the database level of parallelisation, a database of sequences is split into a number of parts or 'fragments', and the same query is searched against each. In [12] and [13] for example, databases are divided into equal fragments on 20 compute nodes, with the analysis extended to many, smaller fragments to achieve better load balancing in the latter case. Additional overhead is incurred in formatting many fragments, which substantially decreases with additional processors, although an optimum value is not deduced.

At query level, the sub-sequences of a single query are examined. Work not based on BLAST is presented in [3] for homologous sequence retrieval, which uses a hybrid 'bucket' method to sort sequences and balance load. A different idea, related to job level parallelisation, is contained in [6], which uses a custom client/server to distribute queries to compute nodes.

Work presented at the level of a single sequence exclusively concerns genomes. In [11], splitting a single sequence into smaller sequences with overlap is contrasted with splitting a sequence into 'seeds' (which are identical to parts of the query sequence and extended to facilitate a similarity search).

In terms of sequence sizes, most investigations looked at average sizes of 1,600 base pairs (bp) [3] or less [8, 10, 12, 13]. An exception [11] considered 940,000bp to be small (in the context of processing a genome), and also analysed medium and large sequences, the latter case constituting a full genome.

In all literature reviewed, examinations at the levels of sequence, query, database and job are exclusive. No hybrid combinations of methods have been implemented, with the exception of [6] and [11], with [11] comparing a hybrid method favourably with a single-level method. In addition, all implementations concern BLAST or similar sequence-search methods. EMBOSS in contrast, contains a wider variety of sequence analysis programs, and this paper seeks to determine if similar parallel methods can be applied across this spectrum.

3 Analysis of EMBOSS Programs

Parallel implementations of BLAST and similar programs are based on splitting data into smaller parts that can be processed in parallel across a number of processors. In order to determine if the same is applicable to programs in the EMBOSS suite, we have first examined the effects on program run time for increasing sizes of sequence data.

3.1 Methodology

EMBOSS programs were run 5 times each with the same sequence on a single machine, and the time averaged to eliminate atypical values. We repeated this a number of times with a larger sequence each time, and plotted the results for execution time vs. size of sequence (Fig. 1). In order to show the effect of larger sequences, only the sequence size was changed, and default program parameters were used.

To implement this, a simple shell script was used that ran all programs in a batch job, recording times in a file. Each run of the batch file represented a single size of sequence, and needed to be run 5 times to average execution time, and a subsequent number of times for larger sequences. A number of programs took a different type of input, such as a nucleic acid sequence, protein sequence, or group of sequences, which were accounted for in the batch script.

Default parameters were defined within EMBOSS programs themselves, and in cases where they were not, example values from the EMBOSS documentation (distributed with the suite) were used. The sequence sizes we used started at 3,000bp (twice the size of the largest 'average' sized sequence used in work cited earlier), and were increased in increments of same, up to 15,000bp.

3.2 Classification

All 160 programs of the suite are classified into 33 groups, viewable either with the 'wosname' program, or on the EMBOSS website [15]. However, a number of programs are duplicated in more than one group, and a number of programs have similar functions (at least in the context of observing their response to larger data sizes). We have chosen to use the defined groupings regardless, in order to give a truer representation of effect across the function groups. However, for the purposes of brevity, we further classify the programs, based on group names in a defined list. These groups are Alignment, Display, Edit, Features, and Proteins.

A number of programs are eliminated from our examination on the basis that they are housekeeping utilities, or do not use either sequences or databases when run.

3.3 Results

The graphs in Fig. 1 detail run times (in log (seconds)) for programs with increasing sequence size (bp). A number of observations can immediately be made. Firstly, in each of the graphs there is a strong concentration of lines with the same trend, bar a few exceptions, telling us that relatively few programs show markedly extreme behaviour as sequence size increases. Secondly, the general trend for all programs is an increase in runtime. Thirdly, Fig. 1(a), Fig. 1(c), and Fig. 1(e) on the left hand side (Edit, Display, and Features respectively) vary in the space 0-1 seconds, whereas Fig. 1(b) and Fig. 1(d) on the right (Protein, and Alignment respectively) have mostly longer runtimes, from 0.1 seconds up to 654 seconds with 15,000bp. Clearly, non-housekeeping applications take

longer (run-times greater than 1.0 seconds). A number of programs in particular have runtimes that are 10-100 times slower than the majority. The slowest 10 are (in ranked order) `inverted`, `est2genome`, `matcher`, `dotmatcher`, `stretcher`, `wordmatch`, `dan`, `palindrome`, `sirna`, and `supermatcher`, all described in [15].

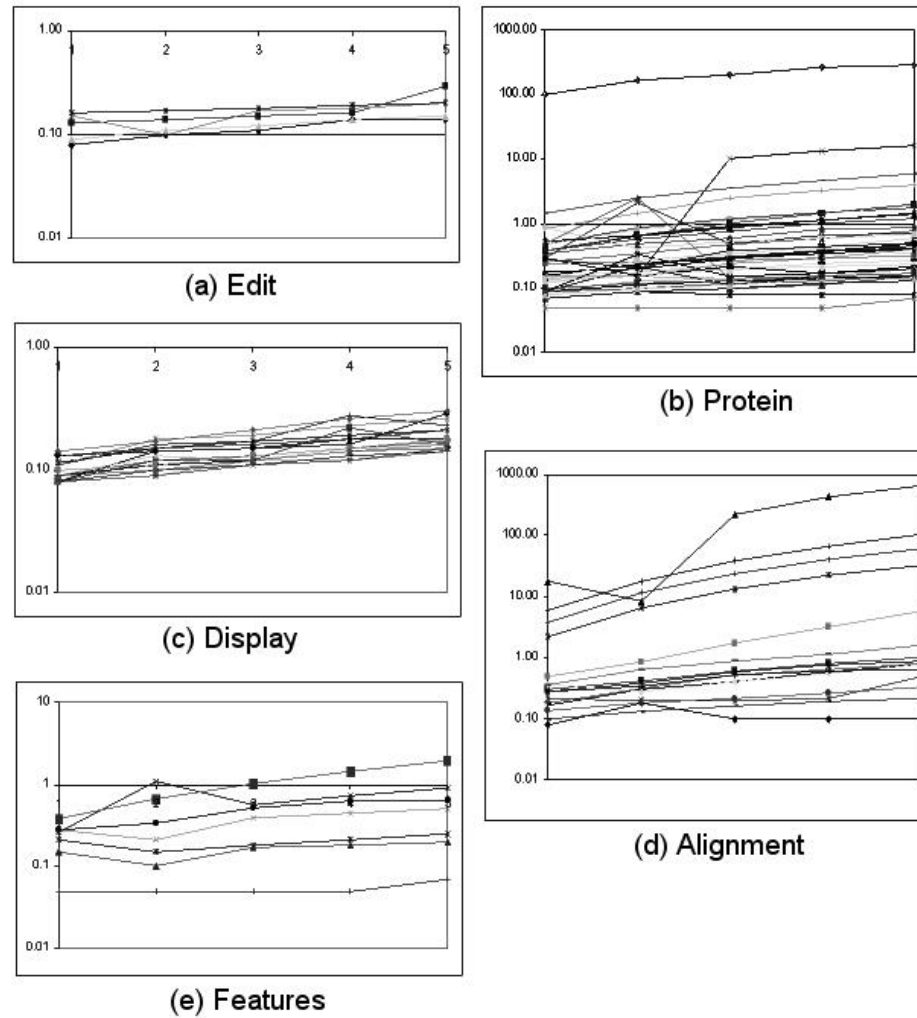


Fig. 1. Execution times for 110 programs of the EMBOSS suite, classified as per Section 3.2. X-axis values are in bp (3000, 6000, 9000, 12000, 15000), and Y-axis values are in log (seconds). The Protein and Alignment groupings contain the most intensive applications, with times in excess of 100 seconds for the largest sequence size

4 Parallelisation of EMBOSS

In this section we introduce a parallel implementation based solely on sequence segmentation, and examine the resulting performance across a cluster of workstations.

4.1 Methodology

For this implementation we adopted a method favoured by a number of previous implementations [10, 14], namely the UNIX script wrapper. This method is fast and easy to develop, and allows program execution to be manipulated without alteration of source code (unlike an alternative method of implementation, MPI [5], which is an API for inserting message passing routines into source code). A disadvantage is that the scripts do not incorporate load-balancing capabilities available in [6], although those capabilities can be added. Strictly, these are not needed to evaluate the raw effect of a parallel implementation, which is of primary concern in this paper.

Using the EMBOSS API, the potential exists for sequences to be split within the suite by individual programs, although this places the burden of input segmentation on each individual program, while patterns may be common across a number of programs.

Using the example of [11], we split a query sequence into smaller sequences with an overlap. To do this we first use an EMBOSS program called 'splitter', which performs the split with a given overlap. Secondly, we remotely copy the smaller sequences to compute nodes using the UNIX utility "rcp". Thirdly, we initiate the analysis on the compute nodes using the UNIX utility "rsh", and finally, use rcp again to collate the results. The execution time required for this entire process is included in the results (Section 4.3), in order to show the scalability of all programs in the EMBOSS suite.

For the data, we chose a large sequence size (relative to the analysis in Section 3), i.e. 60,000bp. At four times the size of the largest data used in Section 3, this facilitates split sizes in a range that we have already examined. We keep this size constant to examine the improvements in performance over different numbers of nodes (the 'scalability'). The relative speedup tells us how fast the program performs compared to a serial run on a single node, and is defined simply as the time taken on n nodes divided by the serial time taken on a single node.

4.2 System

The system used is a modest lab-based cluster of 15 workstations, each with a single PentiumPro running at a clock speed of 180 megahertz (Mhz), 64 Megabytes (MB) of memory, and a 4 Gigabyte (GB) hard drive. Fast Ethernet at 100MB/sec connects the system.

4.3 Results

We illustrate results below on a small selection of programs, randomly taken from the group with the highest execution time (lowest performance). In both graphs (Fig. 2(a) detailing execution times and Fig. 2(b) detailing relative speedup) there is a sharp increase in performance for a small number of nodes, up to approximately 5, where there are minor increases until 12 nodes where performance starts to decrease (e.g. palindrome and sirna). An optimal number of nodes to use can be taken as the lowest value from this interval.

It can be seen from the graphs that performance improvements degrade quicker for programs with shorter execution times, over a large number of nodes. For sirna (shortest execution times), speedup has changed into 'slowdown' over 13 nodes, whereas for einverted (highest execution times), a minor speedup is still maintained. Given that execution times increase for a single program given a larger sequence (as shown in Section 3), there is the potential for speedup to improve for any given program, given a larger sequence size (e.g. genome size).

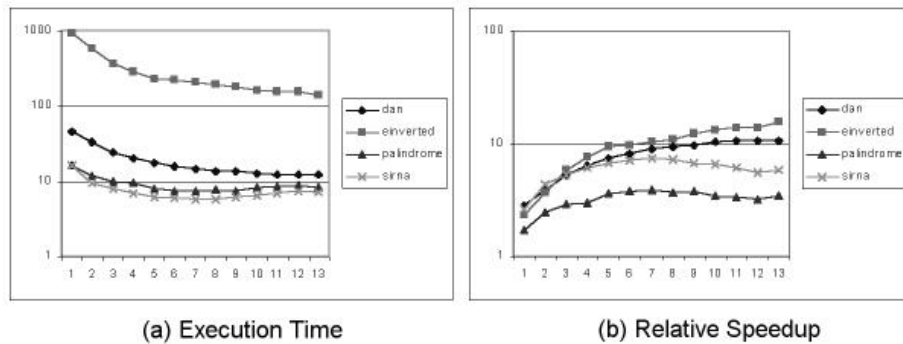


Fig. 2. Results for 4 programs most reactive to sequence size. X-axis values are number of nodes used. Y-axis values for (a) are execution times in log (seconds). Y-axis values for (b) are magnitude of speedup

5 Conclusions

In this paper we have presented both an analysis of 110 programs of the EMBOSS suite and a parallel implementation, based on the variable of sequence size. In the analysis of the suite, we were able to show that the runtimes of all programs were increased for an increasing sequence size. We also showed that the runtimes most affected were in the categories of Alignment and Proteins, whereas tools in Display, Edit, and Features, although affected, were substantially less so. With parallelisation based on sequence segmentation we were able to identify

a plateau of optimum performance across a number of nodes (between 5 and 12, depending on the program used) for a selection of the most intensive applications, and suggest an improvement in program speedup given a larger size of sequence (as shown for programs with larger execution times). Overall, a parallelisation of EMBOSS programs is viable, at least from the point of view of increased sequence size, with major performance improvements possible for a small number of crucial suite applications in particular.

6 Acknowledgements

We would like to thank Dr. A. Bleasby and his group at the RFCGR in Hinxton, Cambridge for introducing the authors to EMBOSS. K. Podesta also acknowledges research support from the National Institute for Cellular Biotechnology (NICB), Ireland, under the HEA PRTLI scheme.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J.: A Basic Local Alignment Search Tool. *Journal of Molecular Biology* 215 (1990) 403-410
2. Fietelson, D.G., Rudolph, L. (Eds.): *Job Scheduling Under the Portable Batch System*. Lecture Notes in Computer Science, Vol. 949, Springer, Berlin (1995)
3. Yap, T.K., Frieder, O., and Martino, R.L.: Parallel Computation in Biological Sequence Analysis. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 3 (1998)
4. Attwood, T.K., Parry-Smith, D.J.: *Introduction to bioinformatics*. Addison Wesley Longman (1999)
5. Gropp, W., Lusk, E., Skjellum, A.: *Using MPI: Portable parallel programming with the Message Passing Interface*. MIT Press (1999)
6. Clifford, R., and Mackey, A.J.: Disperse: a simple and efficient approach to parallel database searching. *Bioinformatics*, Vol. 16, No. 6 (2000) 564-565
7. Rice, P., Longden, I., Bleasby, A.: EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, Vol 16, Issue 6 (2000) 276-277
8. Braun, R.C., Pedretti, K.T., Casavant, T.L., Scheetz, T.E., Birkett, C.L., Roberts, C.A.: Parallelization of local BLAST service on workstation clusters. *Future Generation Computer Systems*, Vol. 17 (2001) 745-754
9. Stevens, R., Goble, C., Baker, P., Brass, A.: A classification of tasks in Bioinformatics. *Bioinformatics*, Vol. 17, No. 2 (2001) 745-755
10. Grant, J.D., Dunbrack, R.L., Mahon, F.J., Ochs, M.F.: BeoBLAST: distributed BLAST and PSI-BLAST on a Beowulf cluster. *Bioinformatics*, Vol. 18, No. 5 (2002) 765-766
11. Vincens, P., Badel-Chagnon, A., Andr, C., Hazout, S.: D-ASSIRC: distributed program for finding sequence similarities in genomes. *Bioinformatics*, Vol. 18, No. 3 (2002) 446-451
12. Mathog, D.R.: Parallel BLAST on split databases. *Bioinformatics*, Vol. 19, No. 14 (2003) 1865-1866
13. Darling, A.E., Carey, L., Feng, W.: The Design, Implementation, and Evaluation of mpiBLAST. *ClusterWorld Conference & Expo and the 4th International Conference on Linux Clusters: The HPC Revolution* (2003)

14. Hokamp, K., Shields, D.C., Wolfe, K.H., Caffrey, D.R.: Wrapping up BLAST and other applications for use on Unix clusters. *Bioinformatics*, Vol. 19 (2003) 441-442
15. <http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Apps/groups.html>. EMBOSS Application Groups. Accessed 13th January 2003
16. <http://www.gnu.org/software/queue/queue.html>. Queue load-balancing/distributed batch processing and local rsh replacement system. Accessed 13th January 2003